REMARKS

Claims 1, 5, 9, 11, 13, and 16-17 are amended. Claims 4, 8, 10, 15, and 20 are canceled without prejudice or disclaimer. No new matter is added by these amendments. Claims 1-3, 5-7, 9, 11-14, and 16-19 are pending. By amending and canceling the claims, applicant is not conceding that the claims are unpatentable over the art cited by the Office Action and is not conceding that the claims are non-statutory under 35 U.S.C. 101, 102, 103, and 112, as the claim amendments and cancellations are only for the purpose of facilitating expeditious prosecution. Applicant respectfully reserves the right to pursue the subject matter of the claims as it existed prior to any amendment or cancellation and to pursue other claims, in one or more continuation and/or divisional applications. Applicant respectfully requests reconsideration and allowance of all claims in view of the amendments above and the remarks that follow.

Rejections under 35 U.S.C. 103

Claims 1-20 are rejected under 35 U.S.C. 103(a) as unpatentable over Bates (US Patent 6,587,967) in view of Schmidt (US Patent Application Publication 2003/0005415 A1) and Akgul (US Patent Publication 2003/0074650 A1). Applicant respectfully submits that the claims are patentable over Bates, Schmidt, and Akgul for the reasons argued below.

Claim 1 recites: "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," which is not taught or suggested by Bates, Schmidt, and Akgul for the reasons argued below.

The Office Action admits that Schmidt does not disclose "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained within any region, and Akgul has no notion of a region and instead bases its decision to stop execution on "if a match is found," which refers to the value of the program counter and the values in the breakpoint registers, so Akgul does not teach or suggest and teaches away from: "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

Claim 1 further recites: "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," which is not taught or suggested by Bates, Schmidt, and Akgul for the reasons argued below.

The Office Action admits that Schmidt does not disclose "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained within any region, and Akgul has no notion of a region, so Akgul does not teach or suggest: "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

Claim 1 further recites: "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program, allowing execution of the first thread to continue after the scoped breakpoint was encountered without giving control to a user," which is not taught or suggested by Bates, Schmidt, and Akgul for the reasons argued below.

The Office Action admits that Schmidt does not disclose "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program, allowing execution of the first thread to continue after the scoped breakpoint was encountered without giving control to a user," as recited in claim 1.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained within any region, and Akgul has no notion of a region, so Akgul does not teach or suggest: "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

Further, Akgul teaches away from "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program, allowing execution of the first thread to continue after the scoped breakpoint was encountered without giving control to a user," as recited in claim 1 because Akgul at column 6, lines 3-4 recites: "only the threads that are marked with breakpoints are interrupted while the other threads are left running." Thus, Akgul requires that threads that are marked with breakpoints are interrupted, which is the opposite of claim 1 where "execution of the first thread [is allowed] to continue after the scoped breakpoint was encountered without giving control to a user."

Further, modifying Akgul to allow threads that have encountered breakpoints to continue execution would render Akgul inoperable for its intended purpose, which is to "[stop] the execution" "[i]f a match is found" between "the value of program counter 904" and "values in the breakpoint registers," as described by Akgul at [0053].

Thus, Bates, Schmidt, and Akgul do not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program, allowing execution of the first thread to continue after the scoped breakpoint was encountered without giving control to a user," as recited in claim 1 because Bates, Schmidt, and Akgul teach away from "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program, allowing execution of the first thread to continue after the scoped breakpoint was encountered without giving control to a user," and no suggestion exists to combine Bates, Schmidt, and Akgul.

Thus, Bates, Schmidt, and Akgul do not teach or suggest all elements of claim 1. Claims 5, 9, 13, and 17 include similar elements as argued above for claim 1 and are patentable over Bates, Schmidt, and Akgul for similar reasons as those argued above. Claims 2-3, 6-7, 11-12, 14, 16, and 18-19 are dependent on claims 1, 5, 9, 13, and 17, respectively, and are patentable for the reasons argued above, plus the elements in the claims. Claims 4, 8, 10, 15, and 20 are canceled without prejudice or disclaimer, so the rejections are moot.

Conclusion

Applicant respectfully submits that the claims are in condition for allowance and notification to that effect is requested. The Examiner is invited to telephone Applicant's attorney (651-645-7135) to facilitate prosecution of this application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 09-0465.

Respectfully submitted,

[illegible]

Date: February 20, 2009

Owen J. Gamon
Reg. No. 36,143
(651) 645-7135

IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

CERTIFICATE UNDER 37 C.F.R. 1.8

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, or is being transmitted via facsimile to the U.S. Patent and Trademark Office, 571-273-8300, or is being transmitted via the Office electronic filing system on: February 20, 2009.

[illegible]

Owen J. Gamon
Registration No. 36,143